2021

Pilot Project

Machine Learning and Content Analytics

MSc in Business Analytics

Dionysios Pavlogeorgatos AM: p2822016
Ilias Tsachtsarlis AM: p2822019

Table of Contents

1. Introduction

This assignment is part of the course Machine Learning and Content Analytics of the MSc program in Business Analytics of Athens University of Economics and Business.

The purpose is to gain hand on experience on the topics instructed during the spring semester of the academic year 2021 related to the core concepts of models, and algorithms from machine learning and content analytics.

Through this report, apart from the understanding of core concepts described above, skills of taking advantage of technologies that processes digital content, (such as documents, news sites, customer conversations, social network discussions a.o), will be enhanced in order to be able to answer specific business questions in a future decision making.

2. Our Project and Vision

The project's goal, named 'pilot project', aims through text sentences processing to identify specific labels than can characterize them.

In specific, having several scientific abstracts, consisted by multiple sentences, linked with the Sustainable Development Goals of the United Nations, our task is to identify and annotate the argumentative statements, for each abstract, giving the appropriate category: claim /evidence.

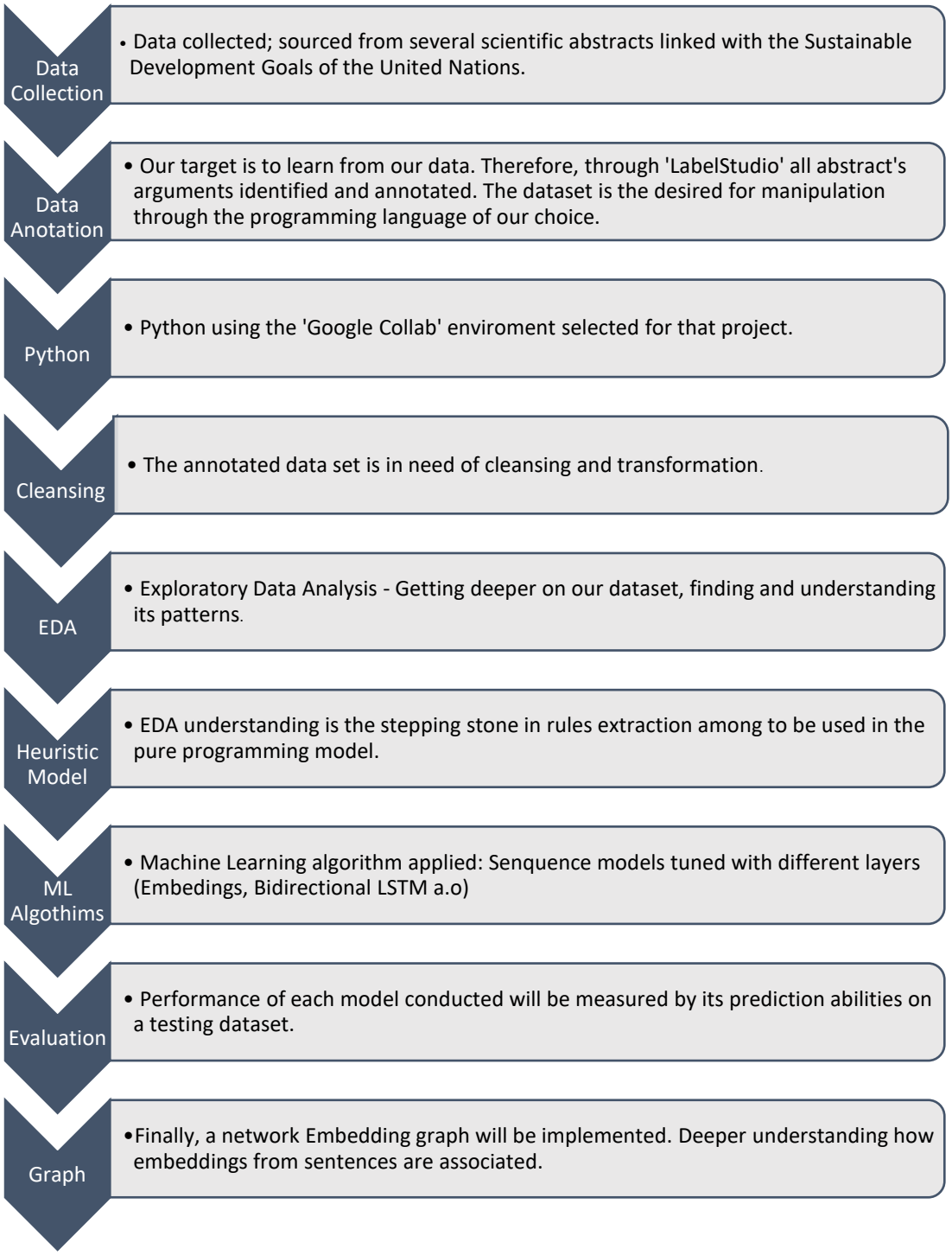
After recognizing the argumentative statements, that knowledge will be taken into advantage in order to create pure programming models and more sophisticated machine learning algorithms, for instance neural networks. These models will be able to predict these argumentative statements given an unrevealed text abstract's sentence comparing to those that model is trained with.

Our vision is to apart from the creation of adequately performing argumentative prediction model, is to get induced in the knowledge path of Machine Learning and

Content Analytics to identify and solve future business problems. More information and details regarding data and methodology can be found on the following chapters.

3. Methodology

In this chapter, methodology is presented below:



4. Data Collection and Annotation

As mentioned in the project description, our dataset consists of 2686 scientific abstracts sourced from the Sustainable Development Goals of the United Nations.

Annotated manually through the ‘Label Studio’ recognizing apart from the ‘Claim’ and ‘Evidence’ arguments also the structural arguments of each abstract such as ‘Background’, ‘Object’, ‘Method’, ‘Result’ and ‘Conclusion’. Since the number of abstracts was quite huge to be annotated, that work got split among other teams in the same project to become timewise. That process is time consuming, needing ten to fifteen minutes per abstract to be fully annotated. Also, the having set timestamps, our work so far was check by comparing the agreement of the annotations for the same abstracts. This agreement is important to distinguish the proper way of annotation, since human judgement differs.

So, let’s explore our dataset using Python programming language…

5. Exploratory Data Analysis (E.D.A)

A first view of our dataset is shown on **figure 1**. Origins from two separated datasets, merges into a 2686 abstracts dataset of three columns. Each row stands for an abstract, named in ‘document’ column, ‘sentences’ column containing in a list form each sentence of the abstract and the ‘labels’ column.

```
dataset_aueb_argument_v3 length: 1017 abstracts
dataset length: 1669 abstracts
data_1 column names: Index(['document', 'sentences', 'labels'], dtype='object')
data_2 column names: Index(['document', 'sentences', 'labels'], dtype='object')

 data_1 dimensions: (1017, 3)
 data_2 dimensions: (1669, 3)
 and the merged dataset dimensions: (2686, 3)
```

	document	sentences	labels
1490	GOQ_G6B3_PMID-25785719.txt	[Maternal mortality in Colombia in 2011: a two...	[NONE, NONE, NONE, NONE, NONE, NONE, NONE, EVI...
1018	DEI_G2B1_23.txt	[Women's economic empowerment, participation i...	[NONE, NONE, NONE, NONE, NONE, NONE, NONE, EVI...
631	doi: 10.1093/ndt/gfx014	[The risk of nephrolithiasis is causally relat...	[NEITHER, NEITHER, NEITHER, NEITHER, NEITHER, ...
1971	RST_G7B4_S0960148113007039.txt	[Evaluating agricultural management practices ...	[NONE, NONE, NONE, NONE, NONE, NONE, EVIDENCE,...
1608	ABC_G1B1_Corpus ID 157686262.txt	[Special Issue of Quantitative Finance on 'Com...	[NONE, NONE, NONE, NONE, NONE, EVIDENCE, CLAIM...

Figure 1 Initial dataset overview

Here to point out that columns ‘sentences’ and ‘labels’ are in correspondence, being each sentence characterized by a label.

The first cleansing action is the conversion of the ‘NONE’ to NEITHER sentences' labels, in order to have a solid structure on the labels’ list elements. Investigating further, the starting sentence of each abstract stands for its title. Since titles are not in our scope, removed, paying attention to remove also the first elements from ‘labels’ list.

Now everything is set to split our dataset into train, test and validation sub datasets distinguishing among ‘X’ and ‘y’, the input variable (or independent variable) and output variable (or dependent variable) accordingly. To be noted that the split made on abstracts level, and a dimensions overview is shown on **figure 2**.

```
X_train shape: (1718,)
y_train shape: (1718,)

X_val shape: (430,)
y_val shape: (430,)

X_test shape: (538,)
y_test shape: (538,)
```

Figure 2 Train, test, validation datasets split among independent and dependent variables.

The ‘train’ dataset will be used in all models building therefore all the EDA will be hold on that. For our purposes its wise not to explore both ‘validation’ and ‘test’ datasets as they will be our key factor on measuring the performance on outer data models are not aware of.

Having a first feeling on labels distribution, a 67% is characterized as ‘neither’, 21% as ‘evidence’ and 12% as ‘neither’ as it is shown on **figures 3 and 4**.

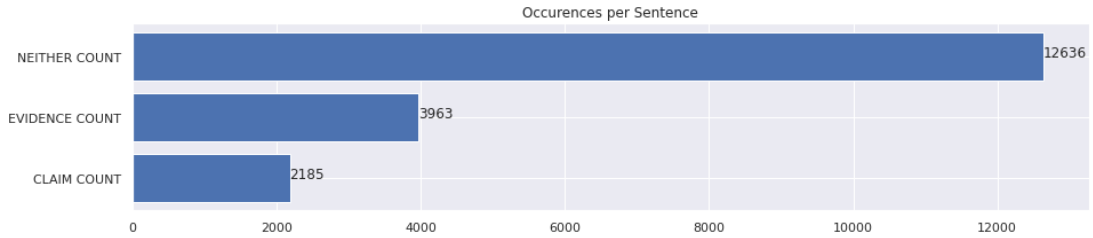


Figure 3 Visual representation of labels distribution – Counts

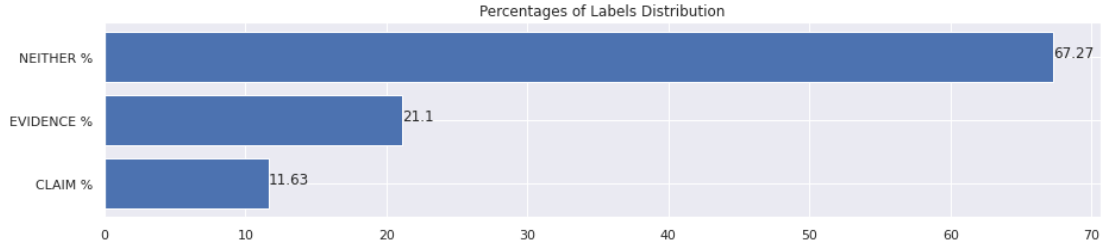


Figure 4 Visual representation of labels distribution – Percentages

Moving forward, unpivoting our dataset on ‘document’ column, and investigating the distribution of sentences per abstract. As it is shown on **figure 5**, most abstracts are not having more than 20 sentences, apart from a few exceptions fluctuating between 40 and 70 sentences.

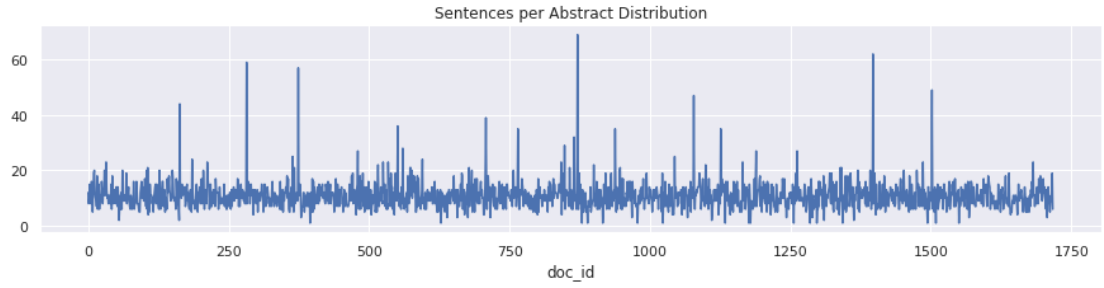


Figure 5 Sentences per abstract distribution

Also, the investigation of the number of words each sentence holds. The distribution presented on the boxplot **figure 6** give as the insight that a median abstract’s sentence consists of 20 words. Also, it is pointed out that now is rare to spot sentences consisted with more than 50 words, but as we show before, the existence of a few long sentences leads to the existence of many words’ cases.

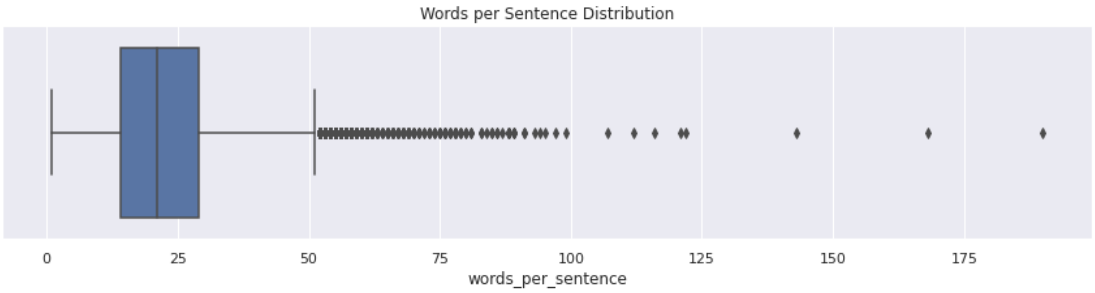


Figure 6 Number of words per sentence boxplot

One step further, investigation on the 'word' level of each sentence follows. That investigation will be held explicitly for the 'evidence' and 'claim' labels separately.

(18784, 5)

	doc_id	sentence	split_sent	words_per_sentence	label
0	0	Abstract	[Abstract]	1	NEITHER
1	0	Objective To investigate the mental health sta...	[Objective, To, investigate, the, mental, heal...	18	NEITHER
2	0	Methods A questionnaire-based survey was condu...	[Methods, A, questionnaire-based, survey, was,...	41	NEITHER
3	0	The 12-item general health questionnaire(GHQ-1...	[The, 12-item, general, health, questionnaire(...	31	NEITHER
4	0	Results The average score of GHQ-12 was (1.09±...	[Results, The, average, score, of, GHQ-12, was...	17	EVIDENCE
5	0	The main mental problems were anxiety and nerv...	[The, main, mental, problems, were, anxiety, a...	8	NEITHER
6	0	Multivariate unconditional logistic regression...	[Multivariate, unconditional, logistic, regres...	78	NEITHER
7	0	Divorce/widowhood (OR=2.351, 95% CI=1.341-4.12...	[Divorce/widowhood, (OR=2.351,, 95%, CI=1.341-...	28	EVIDENCE
8	0	Conclusions The mental health of the floating ...	[Conclusions, The, mental, health, of, the, fl...	13	CLAIM
9	0	However, floating individuals with poor marita...	[However,, floating, individuals, with, poor, ...	21	CLAIM
10	0	The local government should formulate and impr...	[The, local, government, should, formulate, an...	38	NEITHER
11	0	Keywords: 12-item general health questionnaire...	[Keywords:, 12-item, general, health, question...	11	NEITHER
12	1	This study aimed to investigate the regional v...	[This, study, aimed, to, investigate, the, reg...	26	NEITHER
13	1	Data from the 2011 Bangladesh Demographic and ...	[Data, from, the, 2011, Bangladesh, Demographi...	15	NEITHER
14	1	Substantial regional variations in child marri...	[Substantial, regional, variations, in, child,...	10	CLAIM
15	1	Rangpur and Khulna had more than four times hi...	[Rangpur, and, Khulna, had, more, than, four, ...	20	EVIDENCE
16	1	Barisal and Rajshahi had more than three times...	[Barisal, and, Rajshahi, had, more, than, thre...	20	EVIDENCE
17	1	Chittagong and Dhaka had about two times odds ...	[Chittagong, and, Dhaka, had, about, two, time...	28	EVIDENCE
18	1	Respondent's education, employment status, hus...	[Respondent's, education,, employment, status,...	18	NEITHER
19	1	The policy implications of these findings are ...	[The, policy, implications, of, these, finding...	13	NEITHER

Figure 7 New column created, as elements the list of words consisting of each sentence

As it is presented on **figure 7**, everything is set to filter and explore the words that can be considered as labels representatives.

To be noted here some action performed to proceed:

1. words lowering case,
2. word tokenization
3. stop words extraction
4. numerical digits removed
5. manual removal of additional not intuitive words

Evidence:

Having these actions performed by counting the frequency of each word, on **figure 8**, the sorted most common evidence words are presented.

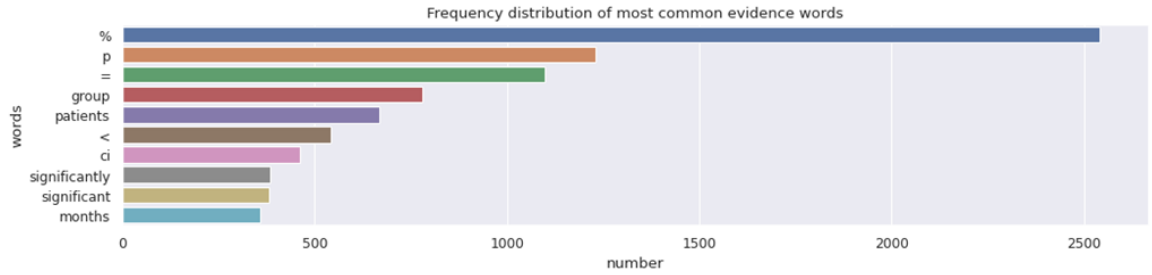


Figure 8 Most common evidence-words

That visual, of the 10 most common words, standing as evidence, frequency distribution is extracted that mathematical equations are commonly used to strengthen a claim. To be noted that the percentage % stands by far the most popular mathematical symbol as it is comfortable and understandable. Also interesting is the place of the letter ‘p’ and ‘ci’ which stand for the ‘p-value’ and ‘confidence interval’ in statistics pointing out if a hypothesis holds or not.

Claims

In the same logic as in evidence, on **figure 9** is shown the ten top common words that appearing on claims.

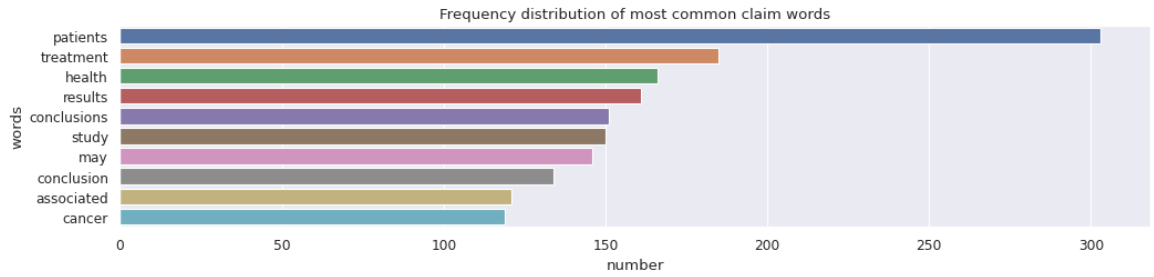


Figure 9 Most common claim-words

That can be commented on claims, is that words like ‘results’, ‘conclusion’ and ‘study’ were expected to be high-placed and therefore impact on the characterization of a sentence as ‘claim’.

Finally, it is quite interesting that the word ‘**patients’** appeared on both ‘claim’ and ‘evidence’ top words. That is a sign of having the most frequent but not the best representators of each group to distinguish among the labels.

For that reason, on the next chapter (*6. Heuristic Model*), by seeking out of the heuristic model’s rules, the lists of ‘evidence’ and ‘claim’ words will be reconstructed.

6. Heuristic Model

By the term of heuristic, we induce the meaning of a model based on pure programming. If some specific conditions hold, a labeling will be decided. So, the core concept is to scan all the sentences of all abstracts one by one, and according to various conditions to decide if are considerable as ‘evidence’, ‘claims’ or ‘neither’.

The first rule constructed inspired from understanding that each label has specific words appearing in large frequency. Though, as mentioned on previous chapter with the word ‘patients’, the most common words are not enough to make the labeling decision as might co-exist on other labels too.

So, our initial move is to keep on both ‘claims’ and ‘evidence’ word-charts the 80% highly appearing words, as shown in **figure 10**, since the rest 20% could be occurred by pure chance.

```
# We  set a threshold at 80%
print(evidence_com_words.number.head(3000).sum() / evidence_com_words.number.sum())
print(claim_com_words.number.head(3000).sum() / claim_com_words.number.sum())

0.820396739213557
0.8236927348449792
```

Figure 10 Threshold set, keeping only the 80% most frequent words.

Now, by removing from both 'claims' and 'evidence' word-lists those words that co-exist, an improved words representation list created for both arguments. An overview of the two unique words top-5 words is shown on **table 1**.

CLAIM	EVIDENCE
Conclusion	Respectively
Safe	p=0
Improves	mm
Design	hg
Efforts	±

The second rule that will be introduced is the claims positioning distribution across each abstract. From the annotation process revealed that claims have pattern to occur most often on the last sentences of each abstract.

This statement eventually holds! Proved that claims are found in the final sentences in possibility of 57%, on second to last with probability 35%, on third to last 15% and on both the last and second to last at 17%. Visually represented on **figure 11**.

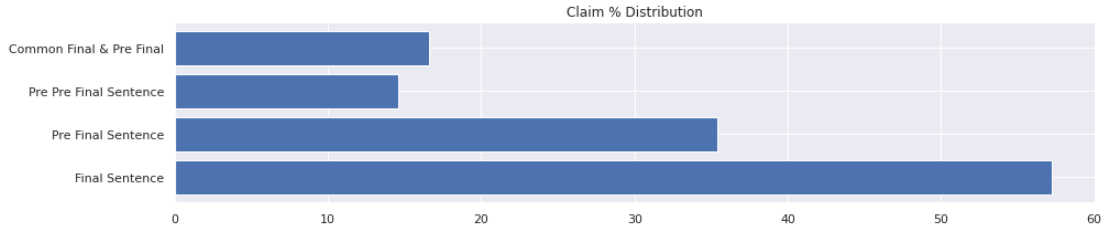


Figure 11 Claims positioning distribution in percentages

The third rule that implemented, is that without the existence of an evidence, a claim cannot be assumed. Therefore, the model will prevent of assigning evidence without any claims detected.

Now, using these 4 rules, heuristic model constructed. The key factor in order to take advantage of the claim's distribution is to start iterating over the the sentences in a reverse order, setting a threshold not to exceed more than 3 claim-sentences per abstract. Model code follows:

```
def baseline_model(data,claim_list,evid_list):

# Empty column initialized to gather the predicted labels
  data['heurclass'] = np.empty((len(data), 0)).tolist()

# Iteration for every abstract
  for i in range(0,data.shape[0]):

    # Counter variables for claims and evidence per abstract
    c = 0
    e = 0

    # Iteration for abstracts sentences in reverse order
    for j in range(-1, -(len(data.sentences[i])+1),-1):

      # Sentences expanded into words
      a = data.sentences[i][j].split()

      # Conditioning deciding each element
      # If a claim word spotted and total claims number do not exceed
      # the 2, then recognized as claim
      if any(i in claim_list for i in a) and c<=2:
        data.heurclass[i].insert(0,'CLAIM')
        c = c + 1

      # If claim exists then searching if the evidence word criteria
      elif any(i in evid_list for i in a) and c>=1 and e<=2:
        data.heurclass[i].insert(0,'EVIDENCE')
        e = e + 1

      # If no claim or evince clue, then assign each sentence as
      # 'neither'
      else:
        data.heurclass[i].insert(0,'NEITHER')
  return data
```

Visual of the labeling results performing in the known dataset is presented on **figure 12**.

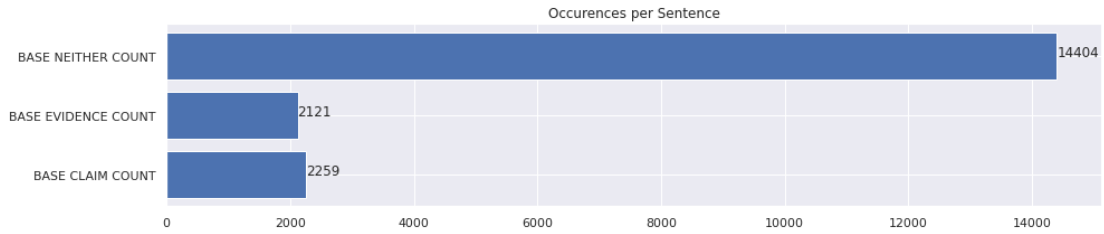


Figure 12 Labeling counts distribution of the heuristic model in the dataset which rules got extracted.

The applied on the training dataset, seem that distribution of 'claim' been kept but regarding the distribution of 'evidence' there is a significant amount that won't get recognized and passed under the label 'neither'.

The truth though will be revealed on the unknown testing dataset, and specifically investigating the f1 precision score.

So, resulting accuracy 58.4% and f1 score at 56.44% lead us to conclude that our baseline needs improvements by building more logical rules.

7. Sequence ML Models

7.1 Data Preparation

Since our dataset is has already been split into train, test and validation datasets, now we proceed with necessary transformations in order acquire the appropriate model sequence format. Overviewing a X and y format in training dataset, **figure 13 and 14**. Respectively the testing and validation datasets formatted as the X and y train.

```
[433]  1 X_train = pd.DataFrame(X_train)
       2 X_train = X_train['sentences'].explode().reset_index().rename(columns={'index': 'doc_id', 'sentences': 'sentences'})
       3 X_train = X_train.sentences
       4 X_train

0        Summary The twin-ATPase ABCE1 has a vital func...
1        As for other functionally distinct ATP-binding...
2        Here, we use an integrated biophysical approac...
3        Our results from FRET experiments show that th...
4        The interaction of ABCE1 with ribosomes influe...
                              ...
18534    Moreover, PM2.5 was also estimated to be respo...
18535    Reducing PM2.5 levels to the EU limit (25 μg/m...
18536    Both scenarios would also attain significant r...
18537    Conclusions:                    Besides its he...
18538    Reductions in PM2.5 concentrations could provi...
Name: sentences, Length: 18539, dtype: object
```

Figure 13 Overview X_train

```
[432]  1 y_train = pd.DataFrame(y_train)
       2 y_train = y_train['labels'].explode().reset_index().rename(columns={'index': 'doc_id', 'labels': 'label'})
       3 y_train = y_train.label
       4 y_train

0         NEITHER
1         NEITHER
2         NEITHER
3        EVIDENCE
4        EVIDENCE
           ...
18534    EVIDENCE
18535    EVIDENCE
18536     NEITHER
18537       CLAIM
18538       CLAIM
Name: label, Length: 18539, dtype: object
```

Figure 14 Overview y_train

7.2 Description of the modelling process

This approach is a more general algorithm that takes as input texts and then captures some of the semantics by placing semantically similar inputs together in the embedding space. Generally, an embedding is a relatively low-dimensional space into which high dimensional vectors are translated. An embedding can be learned and reused across models.

Setting a start point we realized that when you encode something via one-hot encoding, you're making a feature-engineering decision. You're injecting into your model a fundamental assumption about the structure of your feature space. That assumption is that the different tokens you're encoding indeed, one-hot vectors are all orthogonal are all independent from each other to one another. And in the case of words, that assumption is clearly wrong. The embeddings approach is that is most appropriate in our case. That is because similar words get embedded in close locations, and further, specific directions in the embedding space are meaningful. So, word embeddings pack more information into far fewer dimensions in comparison with one-hot encoding which is binary, sparse (mostly made of zeros), and very high-dimensional.

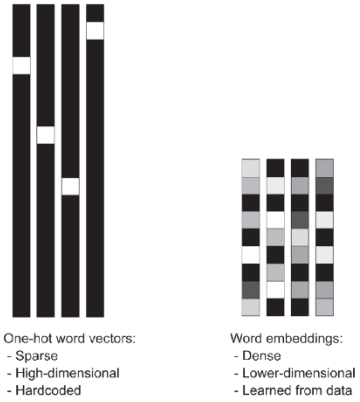


Figure 15 One-hot encoder versus Word embeddings

In our case, one hot encoder of 'sklearn' used on the label encoding as we want to maintain its uniqueness. Overview of that process is shown on **figure 15, 16 and 17**.

```
[439]  1 # At first we run fit_transform on the train dataset
       2 y_train_enc = y_enc.fit_transform(y_train.values.reshape(-1, 1))
       3 y_train_enc

array([[0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       ...,
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.]])
```

Figure 18 y_train Overview

```
[441]  1 # At first we run fit_transform on the validation data
       2 y_val_enc = y_enc.fit_transform(y_val.values.reshape(-1, 1))
       3 y_val_enc

array([[0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       ...,
       [0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.]])
```

Figure 17 y_validation Overview

```
[440]  1 # At first we run fit_transform on the test data
       2 y_test_enc = y_enc.fit_transform(y_test.values.reshape(-1, 1))
       3 y_test_enc

array([[0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       ...,
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.]])
```

Figure 16 y_test Overview

Therefore, firstly we will use the tokenizer that 'Keras' package provides, to take the 22703 most common words of our dataset. All the words are converted to lower case, and we will keep 'out of vocabulary token'. The out of vocabulary token ('OOV') aims to assign a special token to the words that not included to the number of the common words, we have set above. So, in each special token will be given a specific id.

As next step, we fit the tokenizer on the texts of the training dataset, which will generate the tokens by counting the frequency. The creation of sequences intent to assign to each token a specific id. The smaller the number of the id, the more often this word will appear in the dataset. Following, the sequences created above, will be padding because each sentence has different length. So, we set a maximum threshold as far as the number of words of the sentences, let's say 100 in our case and the sequences that are too long will be cut, whereas to those that are too short zeros will be added (pre-padding approach).

Regarding the embeddings model, the 2 most important parameters that we will have to specify, is about the number of words we want to keep having their embeddings and how long the sequences are. Based on empirical rules and the available bibliography, a common practice to find the number of the sequences, is to investigate the distribution of the length of the corpora and then set a threshold that covers the 80% - 90% of all the cases.

Next, we download and loaded the pre-trained glove embeddings. Those used are from Global Vectors for Word Representation (GloVe), which was developed by Stanford researchers in 2014. This embedding technique is based on factorizing a matrix of word co-occurrence statistics. Its developers have made available precomputed embeddings for millions of English tokens, obtained from Wikipedia data and Common Crawl data. The basic logic is that we create a dictionary, and each word corresponds to the pre trained vector containing the embeddings.

However, we should take into consideration the words that are not included in the word embeddings. For those words that do not exist in glove embeddings, we will create random embeddings based on the mean and the standard deviation in order a random normal vector to be created for each word. In our case, the results displayed below:

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882:
  exec(code_obj, self.user_global_ns, self.user_ns)
Embeddings AVG: 0.004451991990208626 | STD: 0.4081574082374573
Found 17207 pre-trained embeddings out of 22703
```

Figure 19 17207 out of 22703 pre-trained embeddings found

We observe that 17207 out of 22703 pre-trained embeddings found, **figure 19**.

Proceeding now on our model built. The main difference of this approach, in comparison with the one with the self-trained embeddings, is that we pass as input in our model the weights which are the embedding matrix we have generated in the previous step. All the other inputs of the embedding layer, the maximum words, the embedding dimension, and the input length are the same. In this layer, no further training performed since we set the respective option (called Trainable) in our model as False. So, we pass in the embedding layer the pre-trained word embeddings and then we freeze this layer and train the rest of the model.

```
[462] 1 model_2 = Sequential()
      2 model_2.add(Embedding(max_words, #22353
      3                       embedding_dim,
      4                       input_length=maxlen, #maxlen
      5                       weights=[embedding_matrix],
      6                       trainable=False))
      7
      8 #model_2.add(Flatten())
      9 model_2.add(Bidirectional(LSTM(64)))
     10
     11 model_2.add(Dense(64, activation='relu'))
     12 model_2.add(Dense(3, activation='softmax'))
     13 print(model_2.summary())
     14
     15 # Notice that we now have far fewer trainable parameters.
     16 model_2.compile(optimizer='adam',
     17                 loss='categorical_crossentropy',
     18                 metrics=['acc'])
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 100, 300)          6810900
_________________________________________________________________
module_wrapper_3 (ModuleWrap (None, 128)               186880
_________________________________________________________________
module_wrapper_4 (ModuleWrap (None, 64)                8256
_________________________________________________________________
module_wrapper_5 (ModuleWrap (None, 3)                 195
=================================================================
Total params: 7,006,231
Trainable params: 195,331
Non-trainable params: 6,810,900
_________________________________________________________________
None
```

Figure 20 Tuning and compiling model.

In general, bigger embeddings vectors, lead to better algorithms. For this reason, in our model, we have chosen embeddings with vector dimension of 300, after also trying 50, 100 and 200. The number of trainable parameters equals to 195.331

In our model, we used the sequential API of 'Keras' to build stacked layers. Also, we added dense layers, with 64 number of neurons. Generally, the bigger the number of neurons, the deeper the network and the more complex the network would be. After that, we used the 'ReLu' activation function. The 'ReLu' activation function will output the input directly if it is positive, otherwise, it will output zero, **figure 21**.

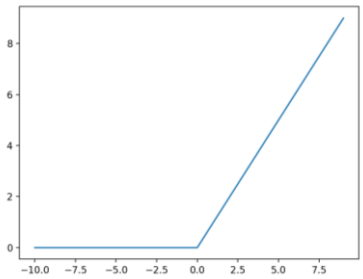


Figure 21 ReLu activation function

The main advantage of using ReLu function in comparison with other functions, is that it does not activate all the neurons at the same time. Consequently, during the backpropagation process, the weights and biases for some neurons are not updated. Furthermore, we used, the 'Softmax' activation function, which is the generalization of the 'sigmoid' activation function. It is important to point out that 'Softmax' always used as final layer output activation function and is used in order to take the inputs and transform them in probabilities.

Regarding the compiling of our model, categorical cross entropy used because in our case we had more than 2 labels. This would be the function, that we would like to minimize during the train process. Even though there are many optimizers available, we chose to use 'Adam' optimizer in our model because is a stochastic gradient descend method and is considered as one of the best adaptive optimizers. Last, metrics option referred in what metrics we would like to observe during training process.

7.3 Results of the Best Model

Having performed many models tuning iterations, investigating the parameters that our sequential model, receives as inputs, we conclude that the smaller number of epochs, in general, lead to better performance in terms of accuracy and f1-score. Also, we observed that the Bidirectional Long Short-Term Memory (LSTM) improves the

performance of our model. The batch size which is a hyperparameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated, equals to 32. Using ReLU and softmax activation functions , our model scored 87% and almost 77% in term of accuracy in training and test datasets respectively. The detailed configuration and the results of our model, presented below, **figure 22- 27**.

*Actual (Rows)/**Predicted (Column)***	**Claim**	**Evidence**	**Neither**
Claim	335	57	267
Evidence	97	587	504
Neither	196	229	3326

Table 1 Confusion Matrix of our best model – Correctly and wrongly predicted labels distibution

```
[2323] 1 model_2 = Sequential()
       2 model_2.add(Embedding(max_words, #22353
       3                       embedding_dim,
       4                       input_length=maxlen, #maxlen
       5                       weights=[embedding_matrix],
       6                       trainable=False))
       7
       8 #model_2.add(Flatten())
       9 model_2.add(Bidirectional(LSTM(64)))
      10
      11 model_2.add(Dense(64, activation='relu'))
      12 model_2.add(Dense(3, activation='softmax'))
      13 print(model_2.summary())
      14
      15 # Notice that we now have far fewer trainable parameters.
      16 model_2.compile(optimizer='adam',
      17                 loss='categorical_crossentropy',
      18                 metrics=['acc'])
```

Figure 22 Model configuration

```
1 history2 = model_2.fit(data_train, y_train_enc,
2                        epochs= 6, #30
3                        batch_size=32,
4                        validation_data=(data_val, y_val_enc))
```

Figure 23 Model Parameters

```
[2328] 1 # evaluate and store on score variable on the TEST DATASET
       2 score = model_2.evaluate(
       3     data_test,    # features
       4     y_test_enc,              # labels
       5     batch_size= 32,   # batch size
       6     verbose=2                # the most extended verbose
       7 )

       181/181 - 8s - loss: 0.6638 - acc: 0.7641
```

Figure 24 Accuracy on testing dataset

	precision	recall	f1-score	support
CLAIM	0.5334	0.5083	0.5206	659
EVIDENCE	0.6724	0.4941	0.5696	1188
micro avg	0.6143	0.4992	0.5508	1847
macro avg	0.6029	0.5012	0.5451	1847
weighted avg	0.6228	0.4992	0.5521	1847

Figure 25 Model F1 recall score

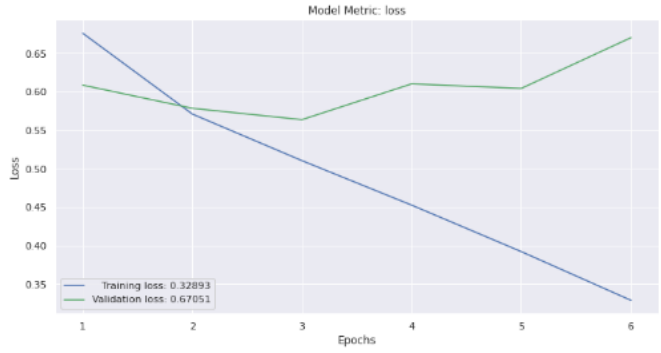


Figure 26 Model Metrics: Loss

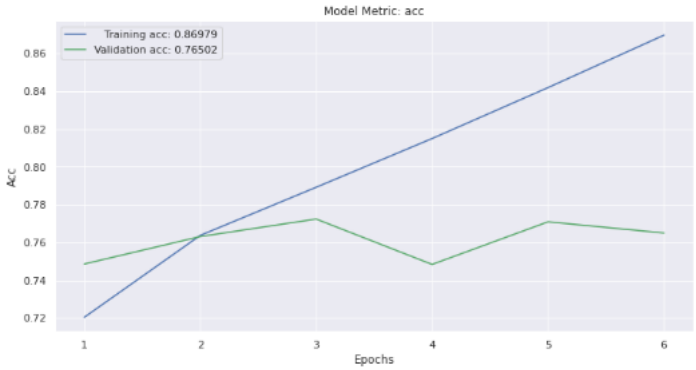


Figure 27 Model Metrics: Loss

7.4 Other Configurations and Results

To find out the model that performs adequately in terms of accuracy and f1 recall, many tuning iterations have been performed. Please find below the configurations and results of three indicative cases.

Starting with a higher number of epochs and Bidirectional Long Short-Term Memory (LSTM) we observed that, with additionally 3 more dense layers and activation functions ReLU and softmax respectively, 15 epochs and 128 batch size, our model scored 97% and 73% in term of accuracy in training and test datasets respectively. This is an example of an **overfitted** model. In other words that model can extensively learn from its trained dataset and perform prediction on it but cannot perform and predict on a testing, out of sample, dataset. Please find relevant **figures 28 - 33.**

1st Model (Overfitted):

```
[2048] 1 model_2 = Sequential()
       2 model_2.add(Embedding(max_words, #22353
       3                       embedding_dim,
       4                       input_length=maxlen, #maxlen
       5                       weights=[embedding_matrix],
       6                       trainable=False))
       7
       8 #model_2.add(Flatten())
       9 model_2.add(Bidirectional(LSTM(64)))
      10
      11 model_2.add(Dense(64, activation='relu'))
      12 model_2.add(Dense(3, activation='softmax'))
      13 print(model_2.summary())
      14
      15 # Notice that we now have far fewer trainable parameters.
      16 model_2.compile(optimizer='adam',
      17                 loss='categorical_crossentropy',
      18                 metrics=['acc'])
```

Figure 28 Model 1 Configuration

```
[2048] 1 model_2 = Sequential()
       2 model_2.add(Embedding(max_words, #22353
       3                       embedding_dim,
       4                       input_length=maxlen, #maxlen
       5                       weights=[embedding_matrix],
       6                       trainable=False))
       7
       8 #model_2.add(Flatten())
       9 model_2.add(Bidirectional(LSTM(64)))
      10
      11 model_2.add(Dense(64, activation='relu'))
      12 model_2.add(Dense(3, activation='softmax'))
      13 print(model_2.summary())
      14
      15 # Notice that we now have far fewer trainable parameters.
      16 model_2.compile(optimizer='adam',
      17                 loss='categorical_crossentropy',
      18                 metrics=['acc'])
```

Figure 29 Model 1 Configuration

```
history2 = model_2.fit(data_train, y_train_enc,
                       epochs= 15, #30
                       batch_size=128,
                       validation_data=(data_val, y_val_enc))
```

Figure 30 Model 1 Fitting Parameter

```
[2053] # evaluate and store on score variable on the TEST DATASET
       score = model_2.evaluate(
           data_test,    # features
           y_test_enc,             # labels
           batch_size= 128,   # batch size
           verbose=2               # the most extended verbose
       )

       46/46 - 7s - loss: 1.2303 - acc: 0.7337
```

Figure 31 Model 1 Metrics: Accuracy on Testing Dataset

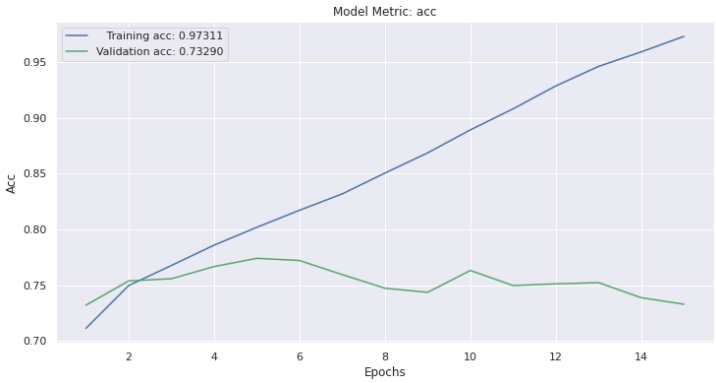


Figure 32 Model 1 Metrics: Accuracy

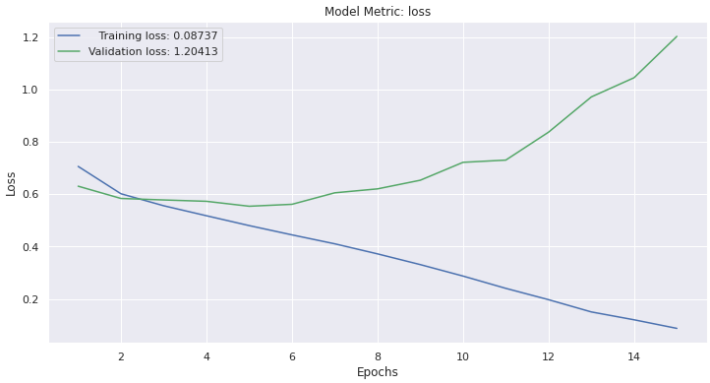


Figure 33 Model 1 Metrics: Loss

2nd Model Results (Ovefitted):

Continuing with a smaller number of epochs and a reduced 64 batch size, without other modifications, our model scored 93% and 73% in term of accuracy in training and test datasets respectively. We observed that the overfitting problem won't get eliminated, and wee need more thing to configure. Please find attached the graphs and metrics for that model on **figures 34 and 35**.

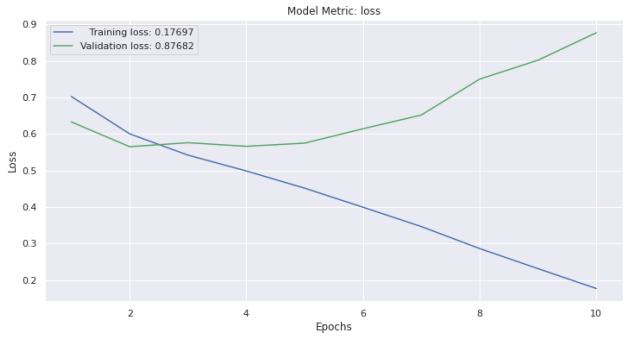


Figure 34 Model 2 Metrics: Loss

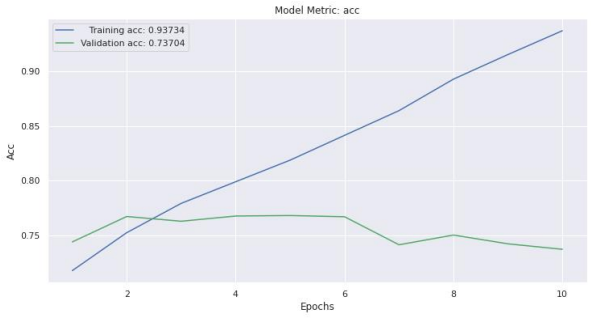


Figure 35 Model 2 Metrics: Accuracy

3rd Model Results:

In the same logic, on the model decided to remove the LSTM layer in order to investigate models' performance on its absence. So, tunning it at 6 epochs, 64 batch size and a 'Ftattern' layer followed by 64 and 3 dense layers, scored 79% and 73% in term of accuracy in training and test datasets respectively. So, concluding again not been that model a good performer, we gain insight on the tunning path we should follow to get the right model. **Figures 36 and 37**.

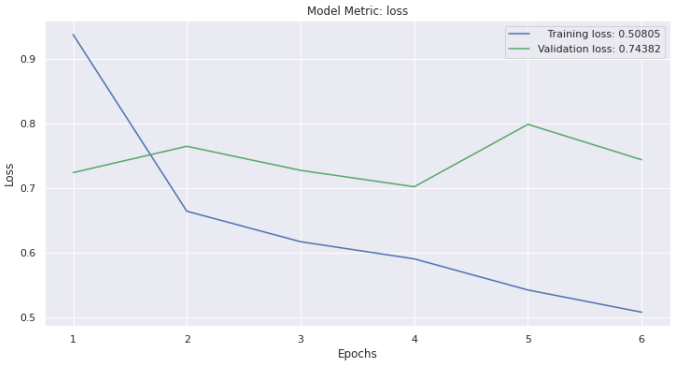


Figure 36 Model 3 Metrics: Loss

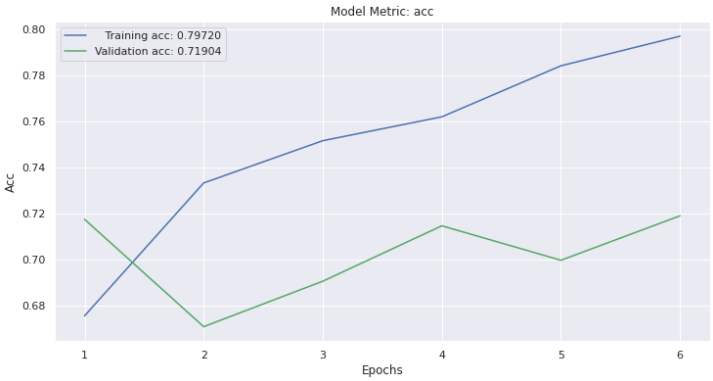


Figure 37 Model 3 Metrics: Accuracy

8. Embeddings Network Graph

Project's final station is the exploration of embeddings in a network graph. As on a great scale explained on previous chapter embeddings are holding a large amount of text information apart from recognizing the exact same words. Can place words and sentences of the same meaning closer in the embeddings space.

Let us focus our investigation on 'claims'. So, all claim sentences per abstract extracted and each embedding produced. The aiming structure, through embeddings, is to find the closest abstracts based on their claim arguments.

Since there are abstracts with more than one claims, decided by computing the averages among those embeddings to proceed created a concrete structure. Now, the Euclidian distances computed, overview of the computation is shown on **figure 38**.

```
1  matrix = pairwise_distances(embeddings, Y=None, metric='euclidean')
2  matrix
```

```
array([[0.       , 5.8435407, 7.273354 , ..., 7.941307 , 7.1675997,
        6.737273 ],
       [5.8435407, 0.       , 5.5930257, ..., 6.908973 , 5.7043266,
        5.2437215],
       [7.273354 , 5.5930257, 0.       , ..., 7.5468016, 6.57909  ,
        5.9219084],
       ...,
       [7.941307 , 6.908973 , 7.5468016, ..., 0.       , 7.8980656,
        7.2727485],
       [7.1675997, 5.7043266, 6.57909  , ..., 7.8980656, 0.       ,
        5.547633 ],
       [6.737273 , 5.2437215, 5.9219084, ..., 7.2727485, 5.547633 ,
        0.       ]], dtype=float32)
```

Figure 38 Abstracts Euclidian distances computed based on claims

Having distances computed, a threshold distance criterion decided, taking into consideration the minimum (0), maximum (9.95) and average (6.42) distances. That criterion determining if two abstracts, based on their claims embeddings distances, are close enough to create and 'edge' between them.

The produced pandas dataframe adjacency matrix produced, having 1 among abstracts and edge exist (close abstracts based on their claims). Overview shown on **figure 39**.

doc_id	0	2	3	4	5	6	7	8	10	11	12	13	15	16	17	18	19	20	21	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	...	2644
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
10	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0
11	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...	0

10 rows × 2413 columns

Figure 39 Pandas dataframe adjacency matrix – 1 declaring sort distance, hence edge

Proceeding the abstract graph created, and a subgraph overview is shown on **figure 40.**

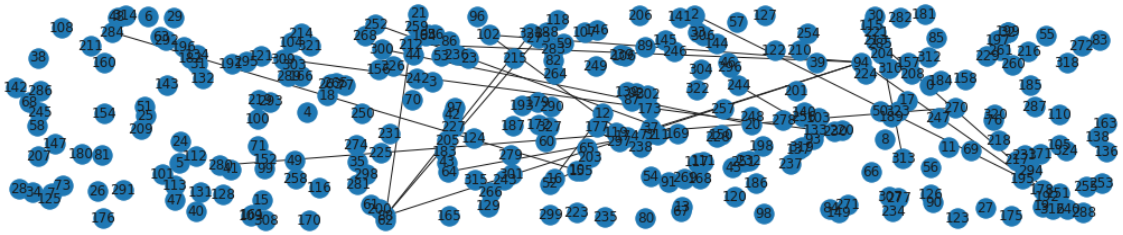


Figure 40 Subgraph overview – Nodes stand for Abstract IDs

Cliques in our graph are strongly associated abstracts based on claims. Therefore, it is interesting to enumerate all of them, and randomly getting deeper in one of them.

```
[333]  1 from networkx.algorithms.approximation import clique
       2 list(nx.enumerate_all_cliques(G))[4000]

       [372, 2422, 2681]
```

Figure 41 Randomly selected clique of 3 – Abstracts IDs portioning that clique extracted

The randomly selected clique, consist of 3 abstracts, with abstracts' IDs shown on **figure 41**.

Moving backwards and extracting their claims we get:

- Claim of 'Doc_Id' 372:

"These results have important implications for policymakers, since the discrepancies in these relationships mean that a country's renewable energy policies should be highly compatible with its development stage."

- Claim of 'Doc_Id' 2422:

'Overall, our results echo other research indicating that non-climate-based frames for renewable energy are likely to garner broader public support, but we suggest that the significance of this finding is particularly important in certain political and geographical contexts.'

- Claim of 'Doc_Id' 2681:

'However, we have identified significant long-term relationships between foreign direct investment, renewable energy consumption and economic growth in some countries, and our study includes important policy implications, particularly for relationship among CO2 emissions and foreign direct investment inflows.'

That can be commented, is that close distance among embeddings reveal the strong association among them, as in all these claims are of the same meaning even if expressed in different words.

9. Conclusion

Through that assignment, apart from being introduced in argumentative prediction model taking advantage of machine learning algorithms with hands on experience, we are able now to identify and solve a future business problem using these in pathway knowledge. Even though our best Sequence model predicts better than the pure programing model (Heuristic), proving the power of neural networks, still there is a a lot of room for improvements and tuning configurations that we can experiment in the future. Finally, we believe learning mission is accomplished and a promising upcoming academic and business path regarding that knowledge gained by the end of the assignment.

Bibliography

[1] Haris Papageorgiou, George Perakis, Aris Fergadis, all class material instructed on Machine Learning and Content Analytics Lectures and Labs, Academic year 202, MSc Business Analytics, Athens University of Economics and Business AUEB.

[2] François Chollet, Deep Learning with Python, Manning Early Access Program (MEAP), Second Edition, Version 7, 2020

[3] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press Ltd, ISBN: 9780262035613, 2017